A Parallel Incompressible Navier-Stokes Solver with a Parallel Multigrid Elliptic Kernel

John Z. Lou*

Abstract

We describe numerical algorithms and parallel implementations of a time-dependent, incompressible Navier-Stokes flow solver and a multigrid elliptic solver which is also used as a computation kernel in the flow solver. The flow solver is based on a second-order projection method (Bell et. al [2]) applied to a staggered finite-difference grid. The multigrid elliptic solver uses the full V-cycle scheme (Briggs, [4]) and was designed to be a general-purpose elliptic solver working on several types of finite-difference grids and for different boundary conditions. A domain-decomposition strategy is used in the parallel implementations for both the flow solver and the multigrid elliptic solver on all tine and coarse grids. The implemented solvers are numerically stable and computational] y efficient, and they scale very well to a large number of processors on Intel Paragon and Cra y T3D for problems with moderate granularity. The solver codes are portable to parallel systems that support MPI, PVM and NX for interprocessor communications.

1.1 ntroduction

The motivation for our work is to develop efficient and reusable parallel partial differential equation (PDF?) solvers that are portable across distributed-memory, message-passing computers, so that these solvers can be used to solve 1 arge, comput at ionall y expensive physics and engineering problems on high-performance parallel computers. A reusable or template PDE solver, in our view, is a PDE solver that can be adapted or expanded to solving a variety of problems using different (component) numerical schemes as needed without a major rewriting of the solver code. For that purpose, we chose a multigrid solver and a fluid flow solver as our testbeds since they are representative of the kinds of numerical schemes and applications we encounter in the field of scientific computing.

The idea of projection method for solving incompressible Navier-Stokes equations was first described in a paper by Chorin [6], which is a finite-difference method for solving the incompressible Navier-Stokes equations in primitive variables. Bell et. al [2] [3] extended the method to second-order accuracy in both time and space, and used a Godunov procedure combined with an upwind scheme in the discretization of the convection term for improved nu merical stability. Projection method is a type of operat or-splitting met hod which separates the solutions of velocity and pressure fields with an iterative procedure. In particular, at each time step, the momentum equations are solved first for an intermediate velocity field without the knowledge of a correct pressure field and therefore no incompressibi lit y condition is enforced. The intermediate e velocit y field is then "corrected" by a projection step in which we

* lou@acadia.jpl.nasa.gov, Jet Propulsion Laboratory, California Institute of Technology, Pasadena, CA 91109

solve a pressure equation and then use the computed pressure to produce a divergence-free velocity field. Our projection step, which is based on a pressure equation derived in [1] and makes use of the highly efficient elliptic multigrid solver we developed, is mathematically equivalent to but different from the projection step described in [2]. In actual flow simulations, this prediction-correction type procedure is usually repealed a few times (2 or 3 iterations seem to be enough from our experiments) until reasonably good velocity and pressure fields have been reached for that time step. In each time step for computing an N-dimensional ($N = 2$ or 3) viscous flow problem using this method, we need to solve m x N Helmholtz equations for the velocity field and m Poisson equations for the pressure field, where m is the number of iterations done at each time step. A fast multigrid elliptic solver is thus very useful to improve the computational performance of the flow solver. The multigrid solver package we developed can solve N-dimensional ($N < 3$) elliptic problems on vertex-centered, cell-centered and staggered grids, and it can deal with Dirichlet, Neumann and periodic boundary conditions.

Since the solvers are implemented on rectangular finite-difference grids, a natural parallel implementation strategy is domain-decomposition: the global computational grid is partitioned and distributed to a logical network of processors; message exchanges are performed for grid points lying on "partition boundary-layer s" (whose thickness is usuall y dictated by the numerical schemes used) to ensure a correct implementation of the sequential numerical algorithms on the global computational grid. In our implementation of the parallel multigrid V-cycle and full V-cycle schemes, we apply this domain-decomposition to all coarse grids as well. 'l'his means on some very coarse grid, only a subset of allocated processors will contain at least one grid point on that grid and therefore they are "active" on that grid, whereas those processors which do not contain any grid point will be idle on that grid, The appearance of idle processors certainly introduces some complexity for a parallel implementation. For example, the logical processor mesh on which the original computational grid is partitioned can not be used for communications on those coarse grids for which idle processors appear. Depending on the type of finite-difference grid and coarsening scheme, one may also need to consider how to correctly apply boundary condition in "boundary processors", which contain at least one grid point next to the boundary of global grid, on those coarse grids, since boundary processors may change from one grid to another. Domain-decomposition on all coarse grids is certainly not the only choice. Another approach, e.g., is to duplicate some of the global coarse grids in every processor allocated, so that evei y processor can do things on these coarse grids independently. The drawbacks of the latter approach include that it involves quite some global communication and it also needs some extra storage for global coarse grids. These requirements may severely affect the sealability of the solver when running on a large number of processors. It is also not obvious at what stage one should duplicate the global coarse grid to achieve a good performance. Although it seems no approach is perfect for implementing a parallel classical multigrid cycle [5] [8], we do believe the use of the domain-decomposition on all grids is an appropriate approach foi implementing a general-purpose parallel multigrid solver. The degradation of parallel efficiency due to the idle processors on some coarse grids has been discussed in many papers (e.g. [5] [8] [9]). The performance measurements from our parallel implementations indicate oui multigrid solver scales quite well on a 512-node Intel Paragon and a 256-node Cray T3D for both 2D and 3D problems with moderate sizes of local finest grid. In fact, the percentage of time spent on those coarse grids is insignificant compared to the total computation time. Similar observation was also made in [8]. As shown by a simple asymptotic analysis in [7], the parallel efficiency of multigrid schemes with the domain-decomposition approach is not qualitatively different from that of a single grid scheme.

The rest of the paper is organized as follows: in section 2, we present numerical algorithms for the multigrid solver and the second-order projection method for the incompressible flow solver; in section 3, we discuss issues related to the parallel implementations of the solvers; in section 4, we show both numerical and parallel performance results from the implemented parallel solvers; section 5 gives some of our observations and conclusions.

2. The Numerical Methods

A. The Multigrid Algorithms

The multigrid schemes we used are the so-called V-cycle and full V-cycle schemes for solving elliptic PDEs, discussed in some detail in [4] and [8]. The full V-cycle scheme is a generalization of the V-cycle scheme which first restricts the residual vector to the coarsest grid and then performs a few smaller V-cycle schemes on all coarse grids, followed by a complete V-cycle scheme on all grids. The full V-cycle scheme often offers a better numerical efficiency than the V-cycle scheme by using a much better initial guess of the solution in the final V-cycle. The parallel efficiency for the full V-cycle scheme, however, is poorer than the V-cycle scheme because it dots more processing on coarse grids. Following shows a pseudocode for the two schemes in a recursive fashion and their graphic representations:

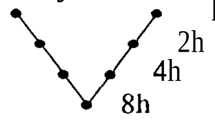
v-cycle Scheme:

```
$V^h \leftarrow MV^h(v^h, f^h)$

1, Relax n times on $A^h u^h = f^h$ with initial $V^h$
2. If $\Omega^h$ = coarsest grid, then go to 4
   else
       $f^{2h} \leftarrow I_h^{2h}(f^h - A^h V^h)$
       $V^{2h} \leftarrow 0$
       $V^{2h} \leftarrow MV^{2h}(V^{2h}, f^{2h})$
3. Correct $V^h \leftarrow V^h + I_{2h}^h V^{2h}$
4. Relax $n_2$ times on $A^h u^h = f^h$ with $V^h$
```

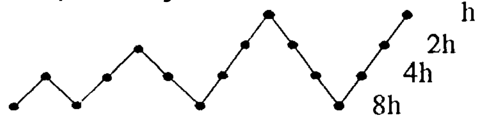
Full V-Cycle Scheme:

```
$V^h \leftarrow FMV^h(v^h, f^h)$

1. If $\Omega^h$ = coarsest grid, then go to 3
   else
       $f^{2h} \leftarrow I_h^{2h}(f^h - A^h V^h)$
       $V^{2h} \leftarrow 0$
       $V^{2h} \leftarrow FMV^{2h}(V^{2h}, f^{2h})$
2. Correct $V^h \leftarrow V^h + I_{2h}^h V^{2h}$
3. $V^h \leftarrow MV^h(v^h, f^h)$
```

V-cycle



full v-cycle



An example: initial grid = 32 x 32

A typical multigrid cycle consists of three main components: relax on a given grid, restrict the resulting residual to a coarse grid, and interpolate a correction back to a fine grid. Our multigrid solver can handle several different types of finite-difference grids commonly used in numerical computations. Figure 1 shows how coarse grids are derived from fine grids for vertex-centered, cell-centered and staggered grids. Although the main steps in a V-cycle are the same for all these grids, restriction and interpolation operators can have different forms on different grids. On a vertex-centered grid we use a full-weighting stencil (9-point averaging on a 2D grid) to make the V-cycle scheme converge well when a point wise red-black Gauss-Seidal (GS) smoother is used; whereas on a cell-centered grid, a nearest-neighbor stencil (4-point on a 2D grid) can be used with the pointwise red-black GS smoother to achieve a good convergence rate. We also point out that, on a vertex-centered grid, the use of the nearest-neighbor restriction stencil with the point-wise GS smoother does not even result in convergence on our test problems, but the use of a Jacobi smoother with the nearest-neighbor restriction stencil results in convergence but with a slower rate. The operator for transferring from a coarse grid to a fine grid is basically bilinear interpolation for all grids. Since fine and coarse grid points do not overlap on cell-centered and staggered grids, one needs to set the values for grid points at the boundary of coarse grids before a bilinear interpolation operator can be applied. More details on the constructions of restriction and interpolation operators for different types of grids can be found in [12].

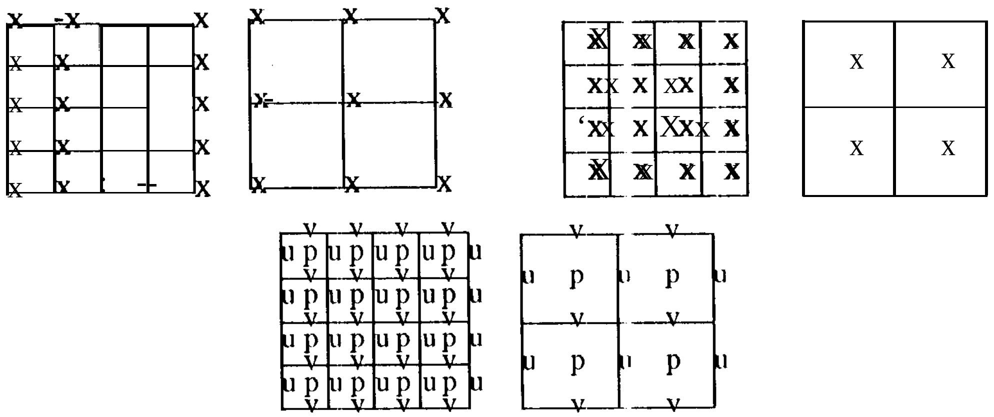
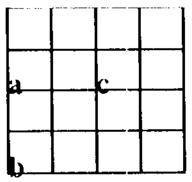


Figure 1: Coarsening of three types of grids: vertex-centered (top-left), cell-centered (top-right) and staggered (bottom).



$$\underset{c}{\frac{1}{16}\begin{bmatrix}1 & 2 & 1\\ 2 & 4 & 2\\ 1 & 2 & 1\end{bmatrix}} \qquad \underset{a}{\frac{1}{16}\begin{bmatrix}0 & 2 & 2\\ 0 & 4 & 4\\ 0 & 2 & 2\end{bmatrix}} \qquad \underset{b}{\frac{1}{16}\begin{bmatrix}0 & 4 & 4\\ 0 & 4 & 4\\ 0 & 0 & 0\end{bmatrix}}$$

Figure 2: Restriction stencils for interior point c and boundary points a and b.

Our multigrid solver can solve Dirichlet and Neumann problems for the grids depicted in figure 1. Periodic boundary condition is also implemented for a special case used in the NS flow solver (to be discussed later). The Dirichlet and Neumann boundary conditions are applied only to the original (finest) grid; a homogeneous (zero) boundary condition is used on all coarse grids since residual equations are solved there. In the case of a Neumann boundary condition, where the unknowns are solved on all grid points including those on the grid boundary, restriction stencils are not well-defined for boundary grid points. Take for example the vertex-centered grid in figure 2, where a 9-point full weighting stencil is used for restriction, This can be done naturally for the interior point c. For boundary points a and b, however, only a subset of the neighboring points are within the grid and therefore weighting stencils on those points still need to be defined in some way. On the other hand, it is reasonable to have the following discrete integral condition satisfied between a pair of coarse and fine grids:

$$\sum_{I,J} U_{IJ} \times A_{IJ} = \sum_{i,j} u_{ij} \times a_{ij} \tag{1}$$

where U_{IJ} and u_{ij} are solutions on coarse and fine grids, and A_{IJ} and a_{ij} are areas of grid cells on coarse and fine grids, respectively. Restriction stencils for interior and boundary points that satisfy equation (1) are given in figure 2.

When solving a Poisson equation with a Neumann boundary condition, the solution is determined up to a constant. We use the following strategy to make sure the application of multigrid cycles converges to a fixed solution: after every relaxation on each grid, we perform a normalization step by adding a constant to the computed solution so that its value at a fixed point (we pick the point located at the center of the grid) is zero. Our numerical tests show this simple step results in a good convergence rate for Neumann problems.

B. The Second-Order Projection Method

We now give a brief description of the second-order projection method for solving the incompressible Navier-Stokes equations in a dimensionless form

$$\begin{gathered} \frac{\partial u}{\partial t} + (u \cdot \nabla) \boldsymbol{u} = -\nabla p + Re^{-1} \Delta \boldsymbol{u} \\ \nabla \cdot \boldsymbol{u} = 0 \end{gathered} \tag{2}$$

where $\boldsymbol{u} \in R^n$ (n = 2 or 3) is the velocity field, $p \in R$ is the pressure field and Re is the Reynolds number. A typical problem is to find u and p satisfying (2) in a domain Ω for a given initial velocity field $\boldsymbol{u}_0$ in Ω and a velocity boundary condition $\boldsymbol{u}_b$ on $\partial\Omega$. The projection method for solving equations (2) is based on the Hedge decomposition which states that any vector field u can be uniquely decomposed into a sum of $U_1 + U_2$ with $\nabla \cdot u_1 = 0$ and $\boldsymbol{u}_2 = \nabla \phi$ for some scalar function ϕ. The projection method proceeds as a type of fractional step method by first writing the momentum equation in (2) in an equivalent form

$$\frac{\partial \boldsymbol{u}}{\partial t} = \mathbf{P} \left(Re^{-1} \Delta \boldsymbol{u} - (u \cdot \nabla) u \right) \tag{3}$$

where $\mathbf{P}$ is an orthogonal projection operator which projects a smooth function onto a divergence-free subspace. Equation (3) can be viewed as the result of applying $\mathbf{P}$ to the momentum equation in (2) which can be rewritten as

$$\frac{\partial u}{\partial t} + \nabla p = Re^{-1} \Delta \boldsymbol{u} - (u \cdot \nabla) \boldsymbol{u}. \tag{4}$$

'I'he projection operation removes the pressure gradient in (4) because ∇p is orthogonal to the projection. Thus if we let the right-hand side of (4) be a vector field V, then $\nabla p = (I - P) V$. The second-order projection method in [2] is a modification to the original projection method proposed in [6] to achieve a second-order temporal accuracy and an improved numerical stability for the nonlinear convection. It uses the following temporal discretization on the momentum equation at each halftime step $n+1/2$

$$\frac{\bar{u}^k - u^n}{\Delta t} + [(u \cdot \nabla) u]^{n+1/2} = -\nabla p^{n+1/2,k} + \frac{1}{Re} \Delta \left(\frac{\bar{u}^k + u^n}{2}\right) \tag{5}$$

where wc assume the velocit y u^n is known, and $\bar{u}^k$ is an intermediate velocit y field that satisfies the same boundary condition as the physical velocity. The discretizations in (5) is second-order accurate in time provided that the convection term can be evaluated to the same order of accuracy at time step $n+1/2$. The superscript k in (5) indicates an iterative process is used for computing u^{n+1}, the velocity at next time step, and $p^{n+1/2}$, the pressure at halftime step $n+1/2$: given a divergence-free field u^n and the pressure filed $p^{n-1/2}$, we set $p^{n+1/2,0} = p^{n-1/2}$ and solve (5) for $\bar{u}^k$. Since the correct $p^{n+1/2}$ is not known, the computed $\bar{u}^k$ is usually not divergence-free; but $\bar{u}^k$ can be used as a guess for u^{n+1} and is used to compute a new guess for $p^{n+1/2}$. Once we have a new guess for $p^{n+1/2}$ it is used in (5) to compute $\bar{u}^{k+1}$. This iterative procedure is repeated at each time step until $p^{n+1/2,k} \to p^{n+1/2}$ and $\bar{u}^k \to u^{n+1}$. In practice, wc found 2 to 3 iterations would be enough to get a satisfactory convergence.

The convection term $(u \cdot \nabla) u$ is evaluated at half time step $n+1/2$ using only the velocity u^n and pressure $p^{n-1/2}$. On the staggered grid shown in figure 1, the pressure p is defined at cell centers, horizontal velocity u and vertical velocity v are defined at cell edges. Let us denote cell (i,j) as the cell whose center is located at $(i - 1/2)\Delta x$, $(j - 1/2)\Delta y$ for $i = 1 \ldots I$ and $j = 1 \ldots J$. $(u \cdot \nabla) u$ is then evaluated at $i, j - 1/2$ for u component and $i - 1/2, j$ for v component. The discretization for u component, for example, has the form

$$[(u \cdot \nabla) u]_u = \frac{u_{i-1/2,j-1/2} + u_{i+1/2,j-1/2}}{2} \left(\frac{u_{i-1/2,j-1/2} - u_{i+1/2,j-1/2}}{\Delta x}\right)$$
$$+ \frac{v_{i,j-1} + v_{i,j}}{2} \left(\frac{u_{i,j} - u_{i,j-1}}{\Delta y}\right)$$

where $u_{i\pm 1/2,j\pm 1/2}$ are velocities at cell centers, $u_{i,j}$ and $v_{i,j}$ are velocities at cell corners and all velocities are assumed to be at time $n+1/2$. Since u^n is the only velocity available at the start of computation for time step $n+1$, we use Taylor expansions of second-order accuracy in both time and space, as was done in [2] and [3], to find velocities at appropriate locations and time for computing the discrete convection term. To improve numerical stability, a Godunov-type procedure combined with an upwind scheme is used in determining velocity values at cell centers and cell corners. To compute u velocit y at the cell center of cell (i,j), for example, we first compute

$$u^R = u^n_{i-1,j-1/2} + \frac{\Delta x}{2} u^n_{x,i-1,j-1/2} + \frac{\Delta t}{2} u^n_{t,i-1,j-1/2}$$
$$u^L = u^n_{i,j-1/2} - \frac{\Delta x}{2} u^n_{x,i,j-1/2} + \frac{\Delta t}{2} u^n_{t,i,j-1/2} \tag{6}$$

where the expansions for u^R and u^L are evaluated on the right side of edge (i -1, j - 1/2) and on the left side of edge (i, j - 1/2), respectively. The choice of $u^{n+1/2}_{i-1/2,j-1/2}$ is then made by the following upwind scheme:

$$u^{n+1/2}_{i-1/2,j-1/2} = \begin{cases} u^R & if \quad u^L > 0,\ u^L + u^R > 0 \\ 0 & if \quad u^L < 0,\ u^R > 0 \\ u^L & othersize \end{cases} \tag{7}$$

The spatial derivatives in (6) are computed by first using a centered differencing and then applying a slope-limiting step to avoid forming new maxima and minima in the velocity field. Temporal derivatives in (6) are computed by using the momentum equation (4). Derivatives at cell corners are computed in a similar way. More details for the constructions of these derivatives are given in [2] and [3].

After evaluation of the convection term, the intermediate velocity $\bar{u}^k$ can be found by solving the following Helmholtz equation for each velocity component:

$$-\Delta\bar{u}^k + \frac{2Re}{\Delta t}\bar{u}^k = 2Re\left(-[(u\cdot\nabla)u]^{n+1/2} + \frac{1}{\Delta t}u^n + \Delta u^n - \nabla p^{n-1/2}\right) \tag{8}$$

We notice that the condition number of the matrix resulted from equation (8) improves as the Reynolds number increases for a fixed grid size and a fixed time step, which is fortunate for computing flows with large Reynolds numbers. For Euler (inviscid) flow problems where $Re = \infty$, $\bar{u}^k$ can be updated explicitly (see equation (5)). Once $\bar{u}^k$ is computed, A projection step is performed to find the pressure $p^{n+1/2,k+1}$ by solving a Poisson equation with a homogeneous Neumann boundary condition

$$\Delta p = R(u^n, u^{n+1}) \tag{9}$$

where $\bar{u}^k$ is used in place of u^{n+1}. Mathematically, equation (9) is the result of applying a divergence operator to the momentum equations in (2). The details for deriving the pressure equation (9) on a staggered grid with appropriate treatments of the Dirichlet velocity boundary condition is given in [1]. In computing a viscous flow, the multigrid elliptic solver is used to solve both equations (8) and (9). After the pressure field is computed, $u^{n+1,k+1}$ can be found by using (5) and this completes one iteration in the next time stepping u^{n+1} and p^{n+1} are then obtained at the end of the last iteration. The flow of control for our incompressible Navier-Stokes solver is shown in figure 3.

3. Parallel Implementations

A. Grid Partition and Logical Processor Mesh

The approach we adopted in parallel implementations of the multi grid elliptic solver and the incompressible flow solver is domain-decomposition. Our objective is to develop parallel solvers that can partition any N-dimensional ($N \leq 3$) rectangular grids and run on any M-dimensional ($M \leq N$) logical processor meshes. For example, figure 4 shows the partition of a three-dimensional grid and the assignment of the partitioned subgrids to a three-dimensional torus processor mesh. As shown in figure 4, logical processor meshes in our code are always constructed as toroidal meshes. Toroidal meshes are useful in the construction of nested

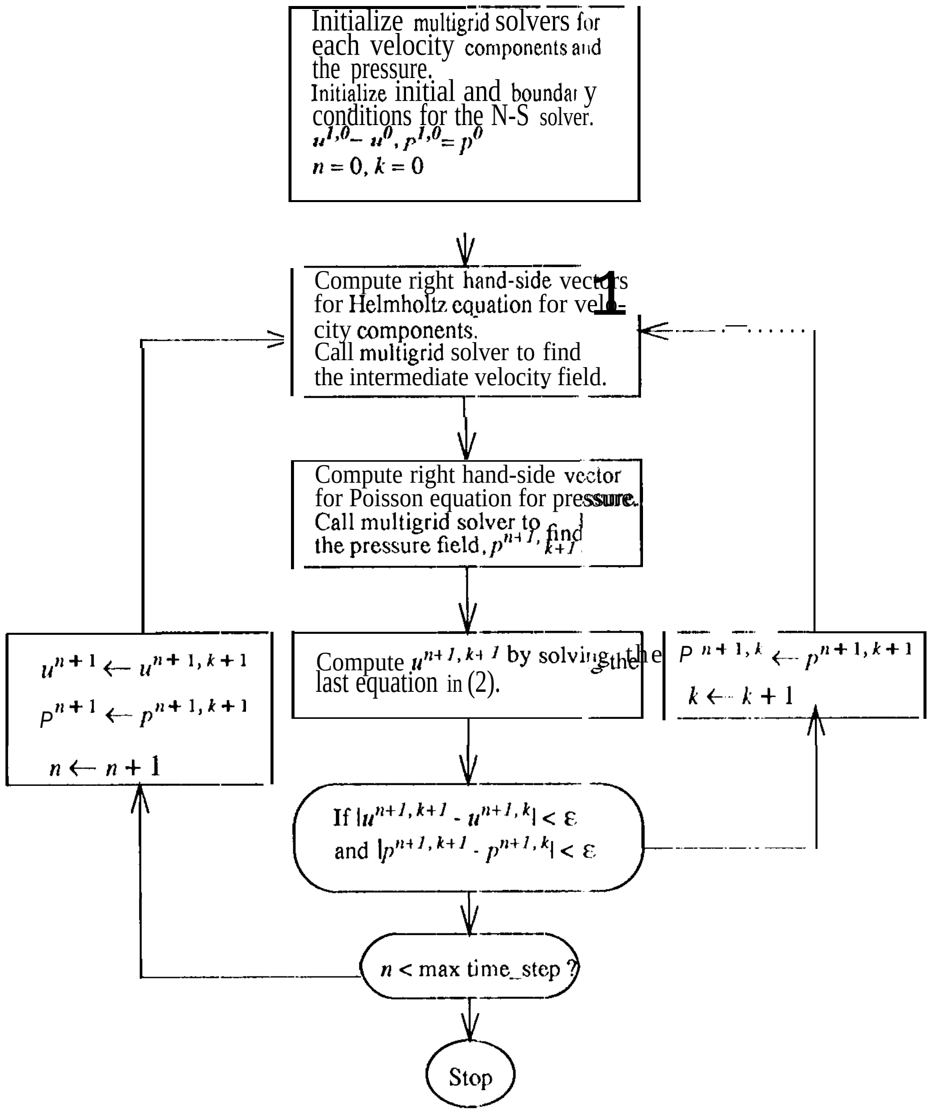

Initialize multigrid solvers for each velocity components and the pressure.
Initialize initial and boundary conditions for the N-S solver.
$u^{1,0} = u^0, p^{1,0} = p^0$
$n = 0, k = 0$
Compute right hand-side vectors for Helmholtz equation for velocity components.
Call multigrid solver to find the intermediate velocity field.
Compute right hand-side vector for Poisson equation for pressure.
Call multigrid solver to find the pressure field, $p^{n+1,k+1}$
Compute $u^{n+1,k+1}$ by solving the last equation in (2).
$u^{n+1} \leftarrow u^{n+1,k+1}$
$p^{n+1} \leftarrow p^{n+1,k+1}$
$n \leftarrow n+1$
$p^{n+1,k} \leftarrow p^{n+1,k+1}$
$k \leftarrow k+1$
If $|u^{n+1,k+1} - u^{n+1,k}| < \varepsilon$ and $|p^{n+1,k+1} - p^{n+1,k}| < \varepsilon$
$n <$ max time_step ?
Stop

Figure 3: Flow diagram for the Navier-Stokes solver

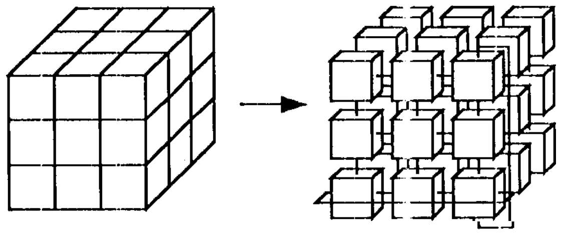


Figure 4: A 3D Grid partition and mapping to a processor mesh, Only two wrap- around connections were shown in the logical processor mesh.

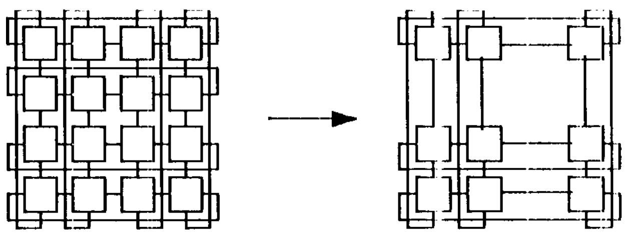


Figure 5: If the left processor mesh contains a 5x5 grid for a Neumann problem on a vertex-centered grid, then the derived coarse. processor mesh is the one on the right.

coarser processor meshes for the mult i grid solver and for dealing with problems with periodic boundary conditions.

In the multigrid solver, coarse grids and coarse logical processor meshes are constructed automatically and recursively based on information on a given fine grid. All grid storages are allocated dynamically during the grid coarsening, process. In particular, for each multigrid level, a local coarse grid is derived from the local tine grid and storages are allocated for the coarse grid. Processors which will get at least one grid point on that coarse grid will be in an active state on that grid, otherwise they will be in an idle state on that grid. A flag is then set in each processor for that level depending on the value of the state. A coarse processor mesh for that coarse grid can then be established by communicat i ng the st atcs among processors in the fine processor mesh, This process is repeated ı cursively until all coarse grids and coarse processor meshes have been constructed. As an illustration, figure 5 shows a processor mesh and its derived coarse mesh for a problem with a Neumann boundary condition. In our multigrid solver, we put this construction process in an initialization routine which must be called before the first time the multigrid solver itself is called. The cost of running the initialization routine is relatively small when one needs to call the multigrid solver a large number of times, as is the case foı the Navier-Stokes flow solver. After executing this initialization routine, every processor knows its "role" at each level of the multigrid cycle, and also knows its neighboring processors on that grid level.

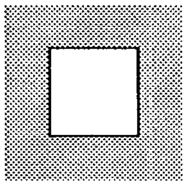


Figure 6: A local subgrid (white area) with surrounding ghost points (shaded area).

B. Interprocessor Communicat ions

To implement the mult igrid scheme and the projection method on a partitioned grid, we need to exchange data which are close to the partition boundaries of each subgrid local to a certain processor, Each processor contains a rectangular subgrid surrounded by some "ghost grid points" which are duplicates of grid points contained in other processors, as shown in figure 6. The number of ghost points on each side of the subgrid depends on numerical algorithms. For the multigrid elliptic solver using a standard Laplacian stencil, one ghost grid point on each side is needed for the local subgrid at each level, whereas for the second-order projection method, three ghost grid points on each side aı e needed in computing the nonlinear convection term using Taylor series and upwind schemes. Therefore in the Navier-Stokes flow solver, wc allocate storages for three ghost grid points foi the fine local grid and one ghost grid point for each coarse grid. For certain operations in the multigrid scheme (e.g. restriction and interpolation) and for computing the convection term in the projection method, ghost grid points in the diagonal neighbor are also needed, as shown in figure 7. Since processors Pi and P_j in figure 7 are not nearest neighbors, direct data exchange between thcm will introduce a more complicated message-passing pattern. Fortunately, direct data exchange between Pi and P_j is not necessary to get the diagonal ghost grid points. It can be verified that all data exchanges that we need are of nearest neighbor types, as indicated in figure 8 for 2D problems. As can be seen in figure 8 that when data 1 ying on partition boundaries are exchanged, the sending blocks always include ghost grid points. After data exchanges in figure 8 are performed, all ghost grid points shown in figure 6 will bc obtained by appropriate neighboring processors. Each processor, therefore, only needs to know its nearest neighboring processors on each logical processor mesh, In solving problems with periodic boundary conditions, data exchanges are also required among processors lying on the boundary of a processor mesh, and the same message-passing operations as shown in figure 8 can bc used.

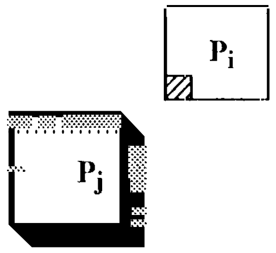


Figure 7: The data in the lower left corner of the subgrid in processor P_i are needed by processor P_j, and stored in P_j's ghost grid points at upper right corner.

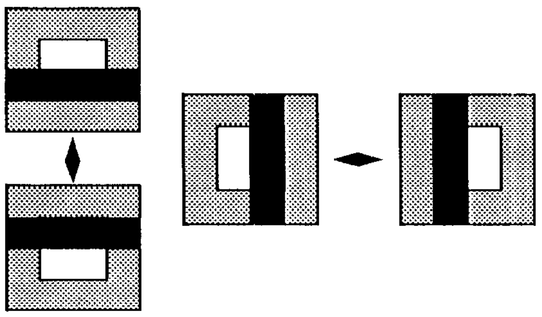


Figure & Data exchanges between neighboi i ng processors for 2D problems. The data in black blocks in each processor are sent out., which is stored in the blocks for ghost grid points in the neighbori ng processors.

The parallel efficiency of a parallel code is largely determined by the ratio of local computations over interprocessor communicati ens. In ou r solvers, the best parallel efficiency is achieved on the finest grid, where the communication cost could bc easily dominated by a large amount of computations, and the parallel efficiency degrades as the grid gets coarser. One way to hide communication overhead and thus improve parallel efficiency on all grids is to overlap communications with computations. In several places within our solvers, we have the following sequence of operations for each processor:

(1) Exchange data lying on partition boundaries;

(2) Perform processing on all local grid points.

'I'o overlap communications with computations, we can perform the following sequence of operations for the same result:

(1) Initiate the data exchange for partition boundaries;
(2) Perform processing on interior grid points that do not need ghost grid points;
(3) Wait until data exchange in(1) is complete;
(4) Perform processing on the remaining grid points.

On Intel Paragon, we implemented the second set of operations above. i n the mult igrid solver and the flower solver using asynchronous message-passing calls. For one full V-cycle in the elliptic solver, for example, the performance improvement on a 256x256 grid partitioned among 256 processors is about 15%, and the improvement on a 256^3 grid partitioned among 512 processors is about 22.%. Faster and asynchronous interprocessor communication can also be achieved on Cray T3D by using its shared-memory communication model, in which direct memory copy is used at either sending or receiving processors for data exchanges between different processors. Somes ynchronization between sending and receiving processors, however, is needed before or after a direct memory copy is performed to ensure the correctness of a message-passing. On T3D processor synchronization is provided only for a group of processors with a fixed stride in their processor indexes, this shared-memory communication model can be easily used for exchange of partition boundary data in the flow solver and for multigrid elliptic solver on some fine grids in which data exchanges only occur between nearest-neighbor processors on the original processor mesh.

C. Soft ware Structures

Our solvers were implemented in C because we think it is the 1 anguage that provides adequate support for implementing advanced numerical software without incurring unreasonable y l arge overhead, Since our goal is to develop reusable. and high-performance PDE solvers which can be used either as library routine or as extensible, template-type code for different applications, we emphasize in our code design both generality and flexibility. First, we want the solvers to be able to run on any M-dimensional rectangular processor meshes for any *N*-dimensional rectangular grids with ($M \leq$ N) (for multigrid processing, N is usually a power of 2). This requirement introduces some complexities in coding the multigrid solver in terms of determining the right global indices for local grids at each grid level. Storages for all grid variables are allocated at run time. For the multigrid solver, storages for local coarse grids are allocated as they are derived recursively from local fine grids. The user is given the option either to supply the storage for variables defined on the or iginal grid or to let the solver to allocate those storages. An array of pointers to an N-dimensional grid (i.e. an N-dimensional data array) is allocated, and each of the pointers points to a grid in the grid hierarchy. *N*-dimensional data array is constructed recursively from one-dimensional data arrays. This strategy of dynamic memory allocation offers a greater flexibility in data structure manipulations and more efficient use of mcrnory than a static memory allocation, and the user is also alleviated from the burden of calculating storage requirements for mtrltigrid processing.

There are two major communication routines in the solvers: the communication routine for the flow solver exchanges partition boundary data only on the original grid; the communication routine for the multigrid solver can exchange partition boundary data for all fine and coarse grids, using a hierarchy of processor meshes. To make the code portable across different message-passing systems, we defined our own generic message-passing library as an interface with our solvers. To use a new message-passing system, we only need to extend the generic message-passing library to that system without changing any code in our solvers. Currentl y, our generic message-passing library can accommodate NX, M PI and WM. A separate data exchange routine has also been implemented for the flow solver, which uses the shared-memory communication library on Cray T3D.

Simple user interfaces to the parallel solvers have also been constructed, The elliptic multigrid solver can be used as a stand-alone library routine with both C and Fortran interfaces. After initializations of the problem to be solved and some algorithm parameters, a preprocessing routine must be called before the first time the multigrid solver routine is called. The preprocessing routine constructs the set of nested grids and the corresponding set of logical processor meshes. The flow solver can be used as a general-purpose incompressible fluid flow solver on a rectangular, staggered finite-difference grid for problems with Dirichlet or periodic velocity boundary conditions. To use the multigrid solver as a kernel for evaluating velocity and pressure fields, the preprocessing routine must be called for each velocity component and the pressure, since they are defined on different grid points on a staggered grid. Therefore separate data structures will be constructed in the preprocessing routine for each velocity component and the pressure, which will be used in subsequent calls to the multigrid solver.

4. Numerical Experiments and Parallel Performances

We now report numerical experiments made to examine the numerical properties of the parallel solvers on a few test problems, and parallel performances in terms of speed-up and parallel scaling of the solvers on Intel Paragon and Cray T3D systems for problems with different sizes and granularities.

A. The Elliptic Multigrid Solver

The multigrid elliptic solver was first tested on Helmholtz and Poisson equations with known exact solutions. Table 1 and table 2 show the convergence rates of the mult i grid solver from solving 2D and 3D Helmholtz equations

$$-\Delta u + u = f$$

with Dirichlet boundary conditions. The runs were performed on Intel Paragon. Errors displayed are the normalized maximum norm of the difference between a computed solution and the exact solution, The tables show the number cycles needed in each case to reach the order of discretization error (or truncation error). At each grid level, two red-black relaxations were performed. Although the full V-cycle scheme is usually considered a lot more efficient than the V-cycle scheme in sequential processing [4], it seems not necessary the case in terms of total execution time cost in parallel processing. As shown in the tables, for the 2D problem, even though the V-cycle scheme takes three more cycles to reach the same order of error than the full V-cycle scheme, the CPU t i me for both schemes are about the same for that test problem; but for the 3D test case in table 2, the full V-cycle scheme is still a little more efficient. With a domain decomposition of both fine and coarse grids, parallel efficiency degrades as the processing moves to coarser grids. Since the full V-cycle scheme dots more processing on coarse grids, its parallel efficiency is worse than the V-cycle scheme. For a large computational grid with many levels of coarse grids, the higher numerical efficiency of the full V-cycle scheme may not improve overall computational performance due to its worse parallel efficiency, as is the case for the 2D case in table 1. For appreciating the effectiveness of the multigrid scheme which has a rate of convergence independent of grid sizes, the errors after a large number of red-black relaxations on the finest grid are also shown in the last rows of the tables. Although not shown in the paper, the convergence rates of the multigrid solver were also measured for cell-centered grid and staggered grid. We found for the same model problem the convergence speeds are slightly slower on those grids, which could be duc to the usc of different restrict ion operators.

Table 1: Numerical Convergence: 2D Helmholtz Solver

Scheme	Grid size	Error	# Processors	# Cycles	CPU sec.
Full V-cycle	2048^2	3.0×10^{-7}	64	3	14.7
V-cycle	2048^2	3.5×10^{-7}	64	6	14.5
Single grid	2048^2	9.7×10^{-1}	64	400 R-B sweeps	124.7

'I'able 2: Numerical Convergence: 3D Helmholtz Solver

Scheme	Grid	Error	# Processors	# Cycles	CPU sec.
Full V-cycle	256^3	1.2×10^{-5}	64	4	71.3
V-cycle	256^3	2.6×10^{-5}	64	8	86.7
Single grid	256^3	8.3×10^{-1}	64	400 R-B sweeps	652.1

The parallel performance of an application code. is usually judged by two measurements: speed-up and scaling, Speed-up is measured by fixing the problem size (or grid size in our case) and increasing the number of processors used. Scaling is measured by fixing the local problem size in each processor and increasing the number of processors used. Although a nice speed-up can be obtained for many applications with a small number of processors, the reductions in CPU time often become very small when a large number of processors is used. This phenomenon is largely inevitable, as stated in Amdahl's law, because as the number of processors used increases for a certain fixed problem size, the communication cost (e.g. the latency for message-passing) and the cost for global operations will eventually become dominant over local computation crest after a certain stage, which makes the influence of a further reduction in local computations very small on the overall cost of running the application. On the other hand, scaling performance of an application seems to be a more realistic measure of its parallel performance, since a code with a good parallel scaling implies, given enough number of processors, it can solve a very large problem in about the same time as it requires for solving a small problem, which is indeed one of the main reasons to use a parallel machine. Figure 9 displays two sped-up plots for a multigrid V-cycle and a full V-cycle for solving 2D and 3D Helmholtz equations wit h a Dirichlet boundary condition on a vertex-centered grid, measured on Intel Paragon and Cray T3D systems. For a comparison, an ideal speed-up curve for one test case is also shown. The code was compiled with the -02 switch on both machines. The grid size for the 2D problem is 512 x 512, and the gi id size for the 3D problem is 64x 64x 64. The maximum number of processors used for the 2D problem is 256 on both machines. For the 3D problem, all 256 processors were used on the T3D (in which case a rect-angular processor mesh of dimensions 4x8x8 was used) and 512 processors were used on the Paragon.

[Figure 9 to be placed here, see it at the end of text]

In terms of single processor performance, we found for our multigrid solver that Cray T3D is about 4 times faster than Intel Paragon, 13ut since the implementation of PVM on T3D, which we used in our code for message-passing, is relatively slow for interprocessor communication, the performance difference for a parallel application on both machines tends to become smaller as granularity of the problem gets finer. We can see for both 2D and 3D problems that the V-cycle scheme has a slightly better speed-up performance than the full V-cycle scheme, which is expected since the full V-cycle scheme does more processing on coarse grids. For the 2D problem, speed-up started to degrade when more than 16 processors were used, and for the 3D problem, the degradation started when more than 8 processors were used, Despite the degradation in speed-up, we can still see some reducti on in CPU time when the largest number of processors was used in each case.

[Figure 10 be placed here, see it at the end of text]
[Figure 11 be placed here, see it at the end of text]

Figure 10 shows the scalings of the parallel multigrid solver on Intel Paragon for problems with three different granularities, using up to 5 J 2 processors. Figure 11 shows the scalings of the same problems on Cray T3D, using up to 256 processors. Shown in the plots are the ratio of CPU times of using n processors versus using one processor. On each of the scaling curves, we fix the local grid size and increase the number of processors, so a flat curve means a perfect scaling. Since a larger global grid has more coarse grid levels for a complete V-cycle or full V-cycle, cost for processing on coarse grids also rises as the number of proces-sors increases, and therefore it has a negative effect on the scaling performance. Like speed-up performance, scaling performance is also largely determi ned by the ratio of local computation cost versus communication cost. This ratio can be dependent on both numerical/parallel algo-rithms and hard ware/software performances on each specific machine. We can see from all the plots in figure 10 and 11 that scaling performance improves as the size of local grid increases. This improvement is expected for an iterative scheme on a single grid, since the computation cost scales as O (n) , where n is the number of grid points in the local grid, whereas the com-munication cost scales as O (n "2'. For a multigrid scheme, it can still be shown that both computation cost and communication cost scale with the same orders as on a single grid [7]. In addition, message-passing latency does not increase as proportionally since the number of messages communicated is still roughly the same for a larger local grid (not exactly the same because more coarse levels are involved) though the size of each message is larger. We can also see V-cycle scheme scales somewhat better than full V-cycle scheme, which is expected since the latter does more operations on coarse grids. The. scaling plots also show 2D test cases scales better than 3D test cases, which we think is due to the fact that 3D grids have a higher surface to volume ratio than the 2D grids and thus the ratio of computations to commu-nications is smaller for 3D cases. As for a comparison between Intel Paragon and Cray T3D, our results show that the scalings on Paragon are slightly better than on T3D. This could be explained by the fact that single processor speed on T3D is much faster than on Paragon, whereas the speed of interprocessor communication on T3D is not proportionally faster when PVM is used for communication.

B. The Incompressible Navier-Stokes Solver

The parallel Navier-Stokes solver was first tried out on a test problem with the following exact solution

$$u = -\cos(x)\sin(y)\,e^{-2t/Re} \qquad v = \sin(x)\cos(y)\,e^{-2t/Re}$$
$$\frac{\partial p}{\partial x} = \frac{1}{2}\sin(2x)\,e^{-4t/Re} \qquad \frac{\partial p}{\partial y} = \frac{1}{2}\sin(2y)\,e^{-4t/Re} \qquad (10)$$

The purpose of the testis to examine the convergence rate of the flow solver, The computation was performed in the unit square $0 \le x, y \le 1$ with initial and (time dependent) boundary conditions specified by the given solution, For numerical stability of the Godunov scheme used in discretizing the convection term, the time step, Δt, *is* restricted by the CFL condition

$$\frac{\Delta t}{\Delta x} U_{max} < 0.5, \qquad (11)$$

where Δx is the size of grid cells and U_{max} *is* the maximum value in the current velocity field. In using multigrid solver for solving velocity and pressure equations, we perform 4 full V-cycles in each call to the elliptic solver. On a 64x64 grid, we found 4 full V-cycles can reduce the residual error to the order of 10^{-10} for the test problem whose solution norm is one, which we think is good enough for our purpose. Table 3 shows the convergence rate of the computed velocity field to the exact velocity field using three Reynolds numbers. The velocity was computed to time= 3.12, which is about 400 time stepson a 64x64 grid, The error in table 3 was measured by

$$e(u) = \|u_{exact} - u_{comp}\|_{max} \qquad e(v) = \|v_{exact} - v_{comp}\|_{max}$$
$$E(u) = max(e(u), e(v)) \qquad Rate = log_2\left(\frac{E(u)^k}{E(u)^{k+1}}\right)$$

where $E(u)^k$ is the error measured on a grid of size $2^k \times 2^k$. A second-order rate of convergence can be seen from the data in table 3. The rate improves slightly as the grid become finer, possibly due to better accuracy on finer grids. The rate also drops slightly as the Reynolds nu mbcr increases, which could be a result of more mr mei ical noise introduced at higher Reynolds number calculations.

Our next numerical experiment on the flow solver is to simulate an evolving 2D driven-cavity flow. The computational domain is still in a unit box $0 < x, y \le 1$. The no-slip velocity boundary condition is applied to all boundaries except at the top boundary, where the velocity value is given, We first test the solver on the problem in which the velocity initial condition is specified by $u = 0$ inside the domain, and the velocity at the top boundary is always one. Figure 12 displays the velocity vector fields which show three stages for time=

0.16, 3.91and 15.63 in the evolution of the flow, computed on a 256x 256 grid with the Reynolds number = 5000. The CFL number (i.e. the right hand-side of (10)) used in the calculation is 0.4, and a total of 10,000 time steps was computed to reach the last state at time = 15.63. Figure 13 shows the vorticity structures at time= 3.91 and 15.63. We found the vorticity structure at time = 15.63 is similar to those obtained by solving the steady incompressible Navier-Stokes equations (e.g. [10]). In running the parallel solver on Cray T3D, the global staggered grid was partitioned and distributed to an 8 x 8 logically rectangular processor mesh, In computing the velocit y vector field, velocity coı nponents defined on cell edges were averaged to the center of cells. For better visibilit y, the vector fields shown in figure 12 are actually 32 x 32 data arrays which were obtained by averaging the 256 x 256 velocity vectors from the simulation. Vorticit y fields were computed at ccl 1 corners by central differencing. The velocity vector plots in figure 12 show clearly how t he cavit y flow develops from its initial state to the final steady state which is characterized by a primary vortex in the center of the unit box and two secondary vortices at the two bottom corners and a small vortex at the upper left corner (e.g. [10]). We also noticed, when reaching the final stage in figure 12, that the change of numerical divergence of the computed velocity field before and after projection is very small. This is because, when the steady state is reached, the intermediate velocit y field would be computed using the correct pressure field to result in a correct velocity field. Even though the initial condition is not continuous along the top boundary and the boundary condi - tion is not continuous at the two upper corners of the unit box, the numerical computation of the flow solver turns out to be quite stable.

Table 3: Second-onkY convergence rate for several Reynolds numbers

Re	64x64	Rate	128^2	Rate	256^2	Rate	512^2
1000	4.628 E-5	1.961	1.195 E-5	1.982	3.010 E-6	2.035	7.568 E-7
3000	2.253 E-5	1.879	6.120 E-6	1.952	1.583 E-6	1.996	3.964 E-7
5000	1.478 E-5	1.824	4.181 E-6	1.845	1.087 E-6	1.870	2.970 E-7

[**Figure 12 to be** placed **here,** see it at the end of text]
[**Figure 13** to **be placed here,** see it at the end of text]

The flow solver was next tried on a driven-cavity flow problem with some smooth initial and boundary conditions. The top boundary now n ıoves with a slip velocity u_t (x) = 16X2 (1 – X^2) and the initial velocity field is specified through a stream function ψ_0 (x, y) = $(y^2 - y^3) u_t$ (.x). The velocity is then computed by $u = -\psi_y$ and v $= \psi_x$. In this case, wc wanted to test the numerical stability of the flow solver on problems with large Reynolds numbers which will result in a very thin boundary layer at the top boundary. Figure 14 displays a velocity vector field and a vorticit y contour from a calculation with Re = 10^5, at time =4.69 for a total of 3000 time steps using a 256 x 256 grid. Figure 15 displays the result from a calculation with Re = 1 0^6 for a total of 7000 time steps on a 512 x 512 grid. These computa -

tions were performed on Cray T3D using 64 processors, We noticed the computations of our solver at these Reynolds numbers seem still numerically stable, which we can judge by checking the convergence rate of the pressure equation and the numerical d ivergence of the computed velocity. The computed flow structures at these Reynolds numbers, however, are quite different from that obtained from the computed flow with Re = 5000. First, at these high Reynolds numbers, the computed flows did not show any sign of approaching a steady state after computing the large numbers of time steps; while with Re = 5000, for the same initial and boundary conditions, wc found a steady state can be reached after computing a smaller number of time steps. Secondly, we can see some interesting flow patterns which do not exist in the flow with Re = 5000. As shown by the vorticity contours in figure 14 and 15, the vorticit y structures in these high Reynolds number flows are much more complicated. We can see a large amount of vortices are generated from the top boundary and then being flushed down along the right wall. Once these vortices reach the neighborhood of the lower right corner, they are pushed toward the interior of the box. We found the vorticit y plot in figure 16 is similar to what reported in [11] where a different algorithm was used on the same problem.

[**Figure** 14 **to be placed here,** see it at the end of text]
[**Figure** 1S 10 **be placed here, see** it at **the** end of **text**]

The second problem we tested on our flow solver is an inviscid fluid flow for which the Eular equations are solved. The computational domain is again restricted to a unit box, and a periodicity of one is assumed in both horizontal and vertical directions. The initial velocity field is given by

$$u = \begin{cases} \tanh (y - 0.25)/\rho & \text{for} \quad y \geq 0.5 \\ \tanh (0.75 - y)/\rho & \text{for} \quad y > 0.5 \end{cases} \tag{12}$$
$$v = \delta \sin (2\pi x)$$

where $\rho = 0.03$ and $\delta = 0.05$. Thus the initial flow field consists of a jet which is a horizontal shear layer of finite thickness, perturbed by a small amplitude of vertical velocity. Since the viscous term is dropped, the velocity can be updated explicitly and the multigrid elliptic solver is only used for solving the pressure equation (9). On the staggered grid we used, the pressure field is defined on a cell-centered grid whose linear dimension, say N, is preferably taken as a power of 2 for the convenience of applying grid coarsening. Thus there are N^2 unknowns for the pressure. Since velocity field is only related to the pressure gradient in the momentum equations, it makes sense to have the velocity defined on an (N -1) x (N -1) grid, as shown in figure 16.

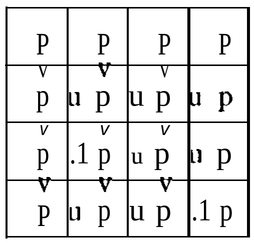


Figure 16: An example of a staggered grid used for computing the doubly periodic shear flow with N = 4. Unknowns for velocit y and pressure in the grid are shown.

Therefore there are (N- 1)2 unknowns for each velocity component. Since the velocity is periodic, a periodic domain should have a dimension of $(N - 1) \times (N - 1)$. Since the pressure gradient is a function of velocit y, it must have the same dimension of periodicity. Thus the physical boundary condition for the pressure equation (9) in the horizontal direction, for example, can be specified as

$$P_{0,j} = P_{1\ j} + P_{N-2,j} - P_{N-1,j} \qquad P_{N+1,j} = P_{N,j} + P_{2\ j} - P_{1\ j}. \tag{13}$$

In a multigrid solution of the pressure equation, the boundary condition (13) is clearly for use on the original, finest grid. The use of condition (13) on any coarse grid, however, is incorrect (our numerical experiments indicate the use of(13) on coarse grids will blowup the computation quickly). Since the unknown vector on a coarse grid is the difference of an exact solution and an approximate solution on the fine grid restricted to that coarse grid, it can be regarded as an approximation to the derivative of the solution on the fine grid. Since a derivative of the pressure field of any order is still periodic with the same period as the velocity field, a reasonable boundary condition for pressure on coarse grids is

$$P_0 = P_N \quad P_{N+1} = P_1. \tag{14}$$

Although condition (14) imposes a period which is one grid cell (of the finest grid) larger than the velocity period, wc found it is easy to apply it to all the coarse grids, and our numerical results show it works well.

Figure 17 shows vorticity contours of two early st ates of the inviscid periodic shear flow. Figure1 8 and 19 show vorticity contours of the flow at time= 1.25 and 2.50, computed from a 128 x 128 grid and a 256 x 256 grid, respectively. The CFL number used in the computations is still 0.4. On the 256x 256 grid, a total of 1600 time steps were computed to reach time = 2.50. These vorticit y plots show how the shear layers, which form t he boundaries of the jet, evolve into a periodic array of vortices, with the shear layer between the rolls stretched and thinned by the large straining field there. A comparison between figure 18 and 19 also shows a better resolution of the vorticit y structure was obtained on the 256x 256 grid.

[Figure17 to be placed here, see it at the end of text]
[Figure 18 to be placed here, see it at the end of text]
[Figure19 to be placed here, see it at the end of text]

The parallel performances of the incompressible flow solver were also evaluated in terms of speed-up and scalabilit y. In each of the parallel performance measurement, we ran the flow solver on the driven-cavity problem for one time step, excluding any initialization and assignment of initial and boundary conditions. Figure 20 shows the speed-up curves of the flow solver on Intel Paragon and Cray T3D systems for three different problem sizes (ideal speed -up curves arc shown again for comparison). The sped-up performance improves as the problem size increases, as expected, For the 512.x512 grid, no significant reduction in execution time could be obtained after more than 64 processors were used. By running the flow solver on a single processor, we found T3D is about five times faster than the Paragon for the code compiled with the -02 switch. But on 256 processors, T3D runs only about 1.5-2.0 times faster than Paragon depending on problem sizes, because, as shown in figure 20, the speed-up performance of the flow solver on Paragon is better than on T3D. Figure 21 shows scaling performances of the parallel flow solver on T3D and Paragon for three local problem sizes. Again, we seethe scaling improves as the size of local grid increases on both machines. In measuring the scalings of the flow solver, we used smaller local problem sizes than we did for the multigrid elliptic solver (see figure 10 and 11). We expect the flow solver to have better scalings than the multigrid elliptic solver because the flow solver, even though calling the elliptic solver several times at each time step, does substantially additional processing on the finest grid, Indeed, this scaling difference between the two solvers can be verified by looking at the scaling curves for the 64 x 64 local grid for the tlow solver in figure 21 and for the multigrid full V-cycle (which is used in the flow solver) in figure 10 and 11. In view of the scaling performances in figure 21, we would claim that our parallel flow solver scales quite well on large numbers of processors as long as the local grid size is not smaller than 64x 64.

[**Figure 20** to **be placed here, see** it **at the** end 01 text]
[**Figure** 21 **to be placed** here, **see** it at the end of text]

5. **conclusions"**

In this paper we presented multigrid schemes for solving elliptic PDEs and a second-order finite-difference projection method for solving the Navier-Stokes equations for incompressible fluid flows. Our parallel implementation strategies based on domain-decomposition arc discussed for implementing these algorithms on distri buted-memory, massivel y parallel computer systems. Our treatment of various boundary conditions in implementing these parallel solvers are also discussed. We designed and implemented these solvers in a highly modular approach so that they can be used either as stand-alone solvers or as expandable template codes which can be used in different applications. Several message-passing protocols (MPI, PVM and Intel NX) have been coded into the solvers so that they arc portable to systems that support one of these interfaces for interprocessor communications.

Numerical experiments and parallel performance measurements were made on the implement ed solvers to check their numerical properties and parallel efti ciency. Our nu meri - cal results show the parallel solvers converge with the order of numerical schemes on a few test problems. Our numerical experiments also show the flow solver is stable and robust on viscous flows with large Reynolds numbers as well as on an inviscid flow. Our parallel efficiency tests on Intel Paragon and Cray T3D systems show that good scalability on a large number of processors can be achieved for both the multigrid elliptic solver and the flow solver as long as the granularit y of the parallel application is not too small, which wc think is typical for applications running on distributed-memory, MIMD machines. For future work, we plan to extend the flow solver to 3D problems and explore a finite-elements implementation of the projection method for solving problems in unstructured domains.

Acknowledgments:

I want to thank Dr. Robert Ferraro (Jet Propulsion Laboratory) for his encouragement and support on this work, 1 would also like to thank Dr. Sefan Vandewalle (California Institute of Technology) and Dr Steve McCormick (University of Colorado) for some helpful discussions on multigrid methods. This work was carried out at the Jet Propulsion Laboratory (JPL), California Institute of Technology (Caltech), under a contract with the National Aeronautics and Space Administration (NASA) and as a part of the NASA High-Performance

Computing and Communications for Earth and Space Sciences Program. The computations were performed on Intel Paragon parallel computers operated by JPL and by the Concurrent Supercomputing Consortium at Caltech, and on the Cray T3D parallel computer operated by JPL.

References:

1. C. Anderson, "Derivat ion and Solution of the Discrete Pressure Equations for the Incompressible Navier-Stokes Equations," Lawrence Berkeley Laboratory Report, LBL-26353, 1988, Berkeley, CA

2. J. B. Bell, P. Colella and H. Glaz, "A Second-Order Projection Method for the Incompressible Navier-Stokes Equations," J. Comp. Phys., 85:257-283, 1989

3. J. B. Bell, P. Colella and 1,.11, Howell, "An Efficient Second-Order Projection Method for Viscous incompressible Flow." Proceedings, 10th AIAA Computational Fluid Dynamics Conference, Honolulu, HI, pp.360-367, 1991

4. W. Briggs, "A Multigrid Tutorial," SIAM, Philadelphia, 1987

5. T. F. Chan and R. S. Tuminaro, "A Survey of Parallel Multigrid Algorithms", in "Parallel Computations and Their Impact on Mechanics", A. Noor, Ed., Vol: AMD 86, 1986

6. A. J. Chorin, "Numerical Solution of the Navier-Stokes Equations," Math. Comp., vol. 22, pp. 745-762, Gel. 1968.

7, G. Fox, et. al., "Solving Problems on Concurrent Processors." Vol. 1, Prentice Hall, Englewood Cliffs, New Jersey, 1988

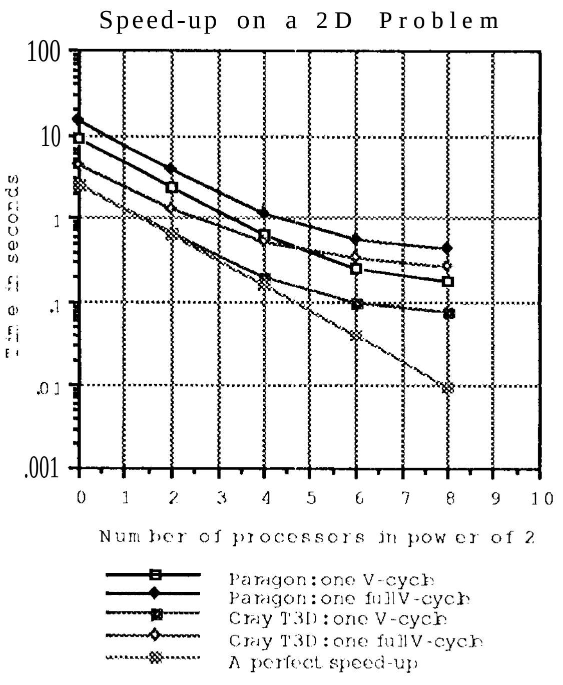
8. S. F. McCormick, "Multilevel Adaptive Methods for Partial Different ial Equations." Frontiers in Applied Mathematics, SIAM, Philadelphia, 1989

9. F. Roux and D. Tromeur-Dervout, "Parallelization of a Multigrid Solver via a Domain Decomposition Method." Manuscript, 1994

10. R. Schreiber and H. B. Keller, "Driven Cavit y Flows by Efficient Numerical Techniques." J. Comp. Phys., 49,310-333, 1983

11. Weinan E and Jian-Guo Liu, "Essentially Compact Schemes for Unsteady Viscous Incompressible Flows.", manuscript.

12. P. Wesseling, "An Introduction to Multigrid Methods", Pure & Applied Mathematics, A Wiley-Interscience Series of Texts, Monographs& Tracts, John Wiley& Sons, 1991



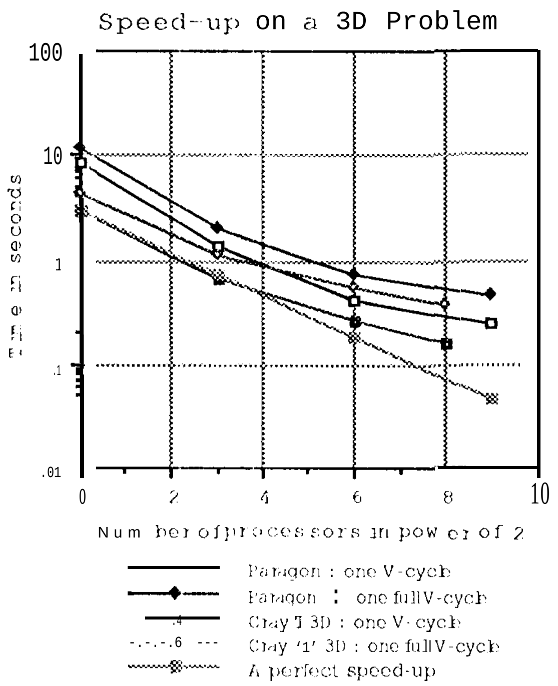


Figure 9: Speed-up performances of the elliptic multigrid solver,

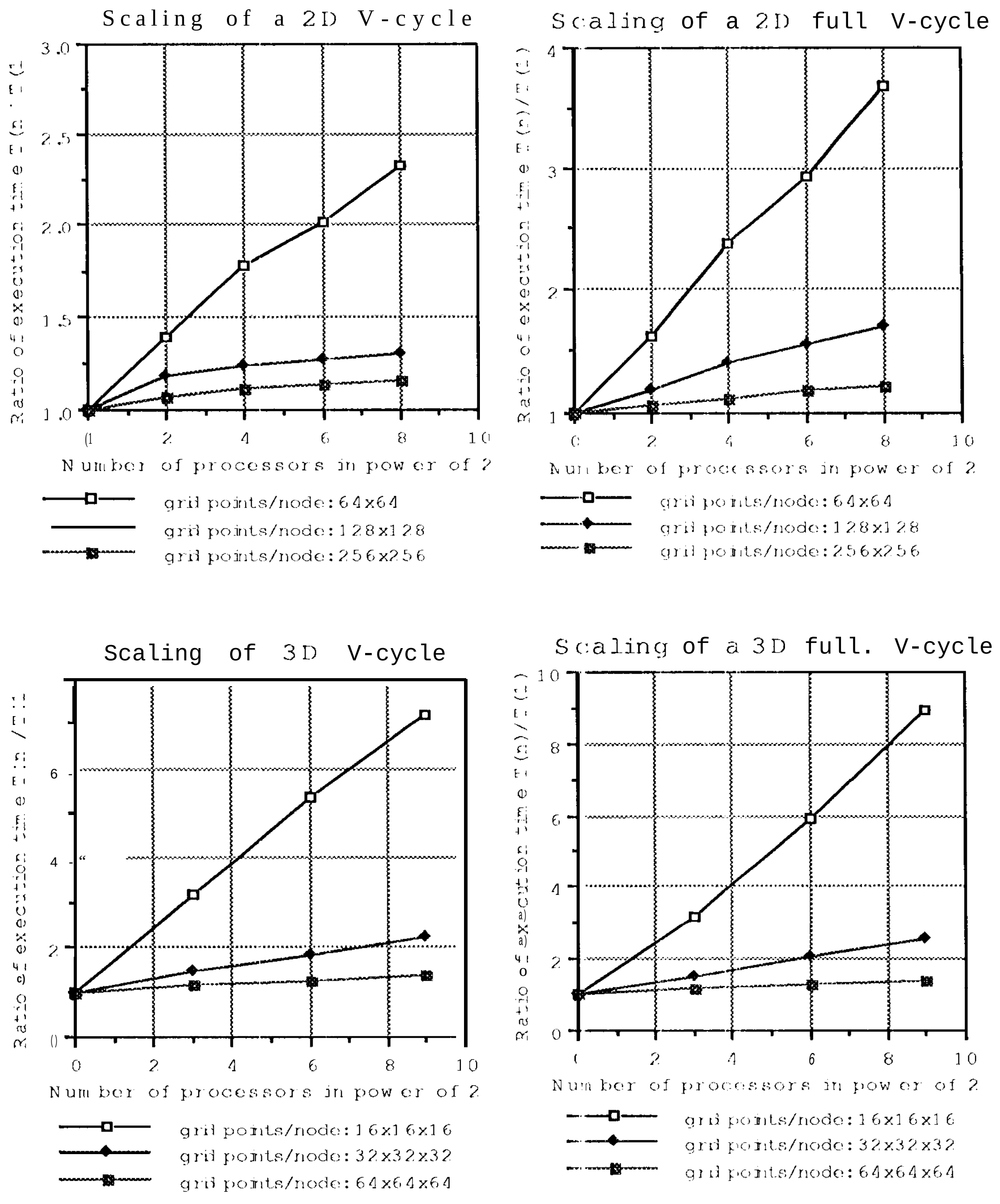


Figure 10: Scaling of the multigrid solver on Intel Paragon.

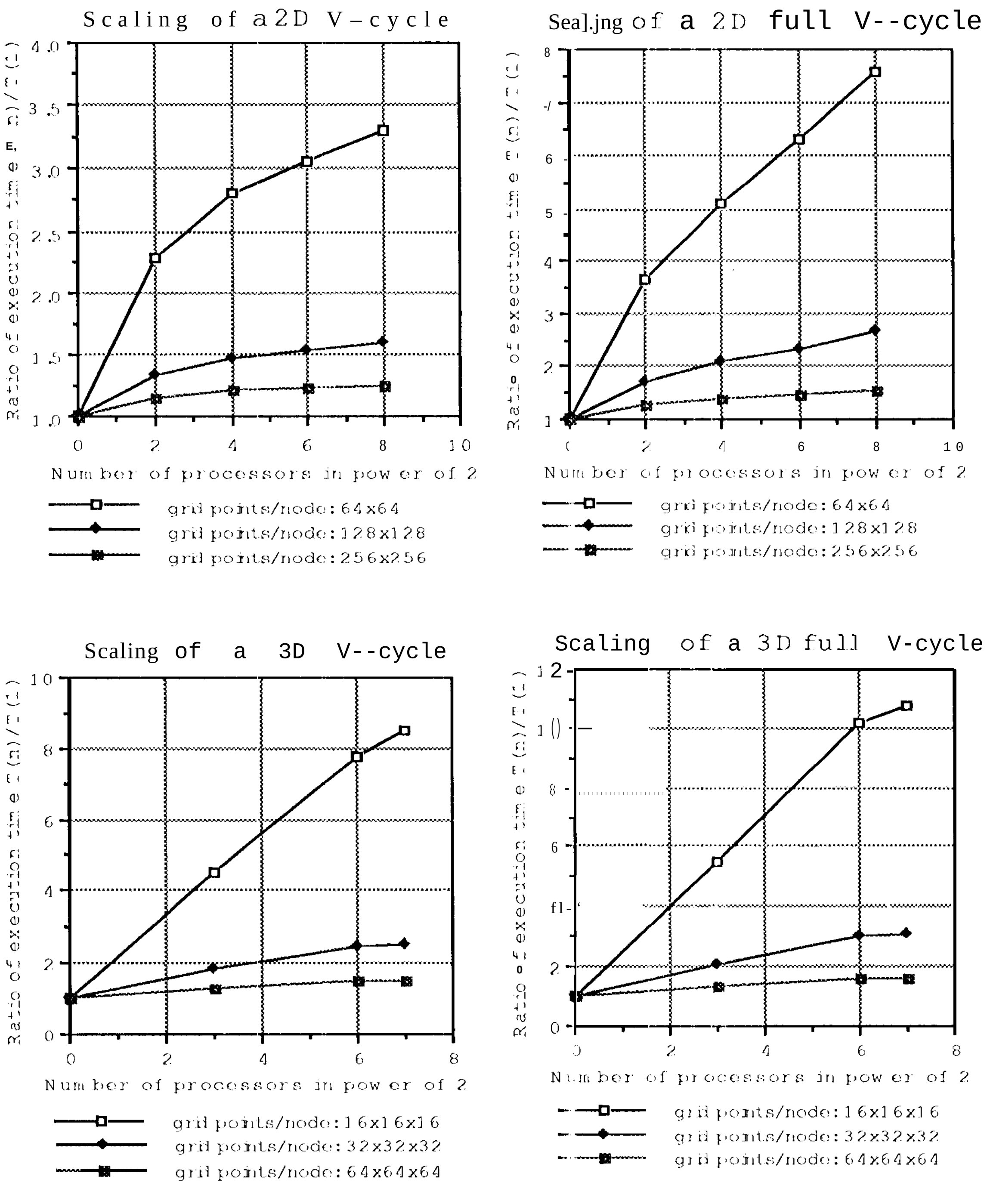


Figure 11: Scaling of the multigrid solver on Cray T3D.

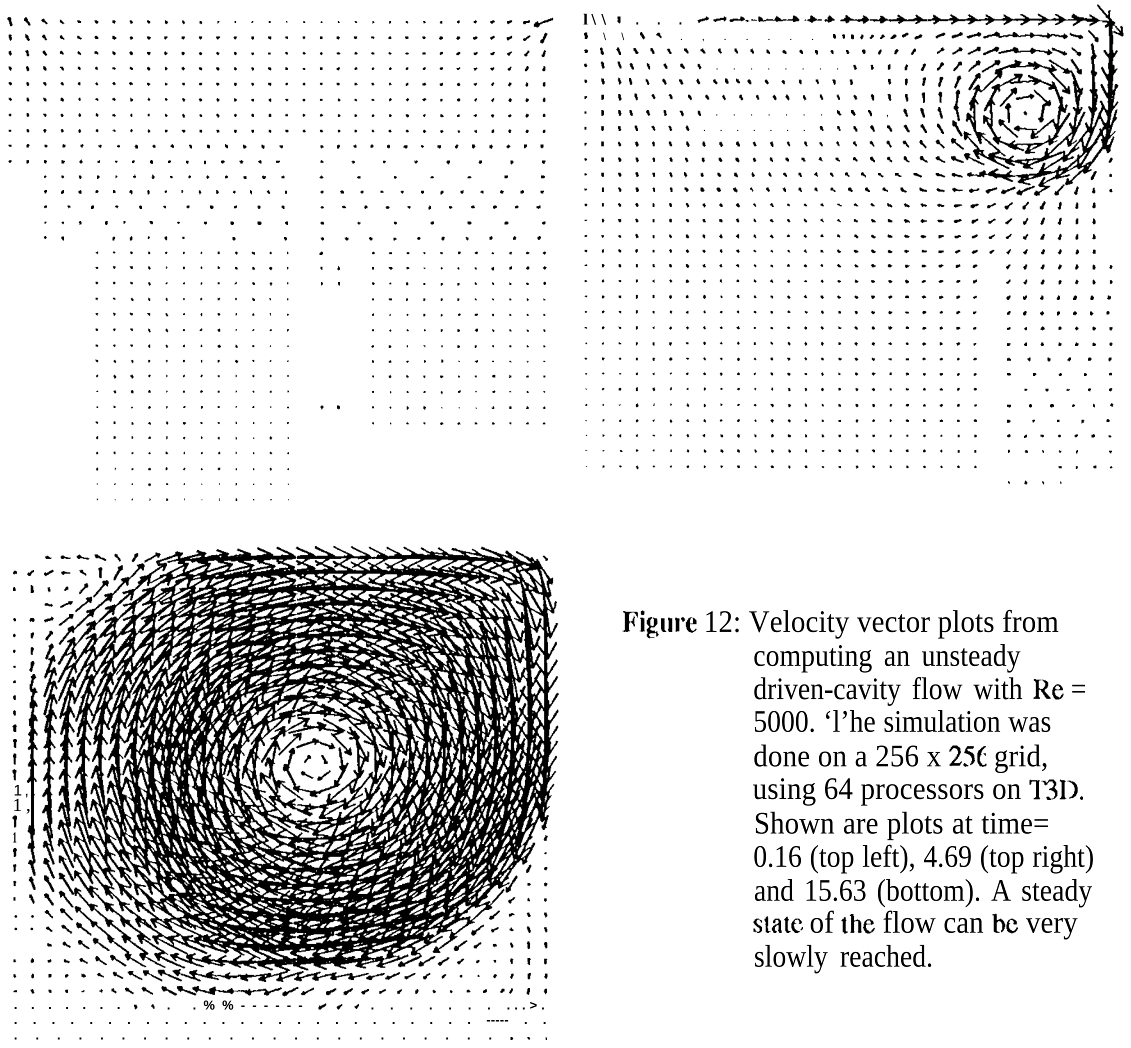
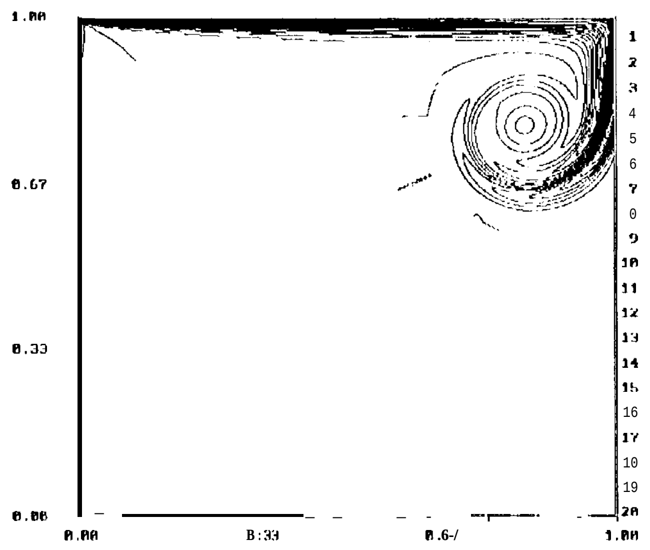


Figure 12: Velocity vector plots from computing an unsteady driven-cavity flow with Re = 5000. The simulation was done on a 256 x 256 grid, using 64 processors on T3D. Shown are plots at time= 0.16 (top left), 4.69 (top right) and 15.63 (bottom). A steady state of the flow can be very slowly reached.



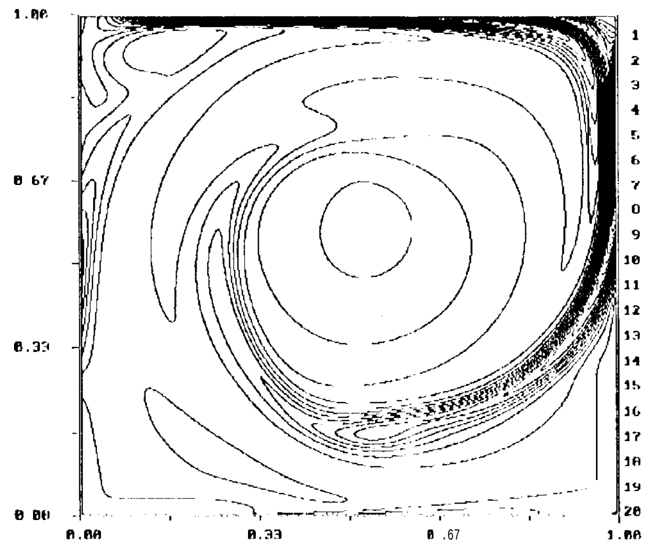
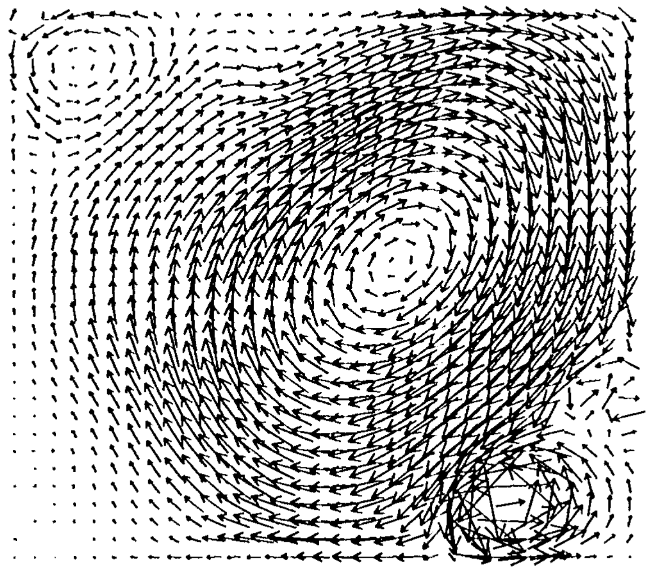


Figure 13: Vorticity contour plots from a driven-cavity flow with Re = 5000, at time = 3.91 (top) and time = 15.63 (bottom). Grid size= 256x 256.



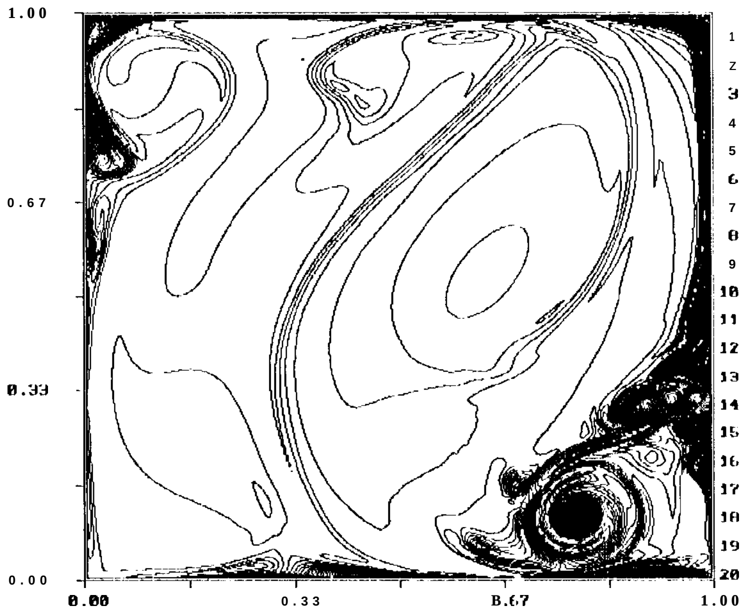
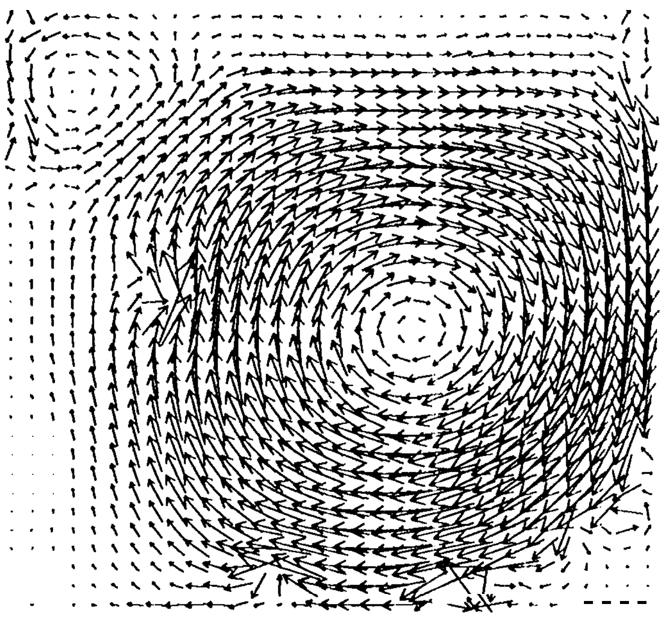


Figure 14: Velocity vector field (top) and vorticity contour plot (bottom) from the driven-cavity flow with Re = 10^5, at time = 4.69. Grid size = 256x 256.



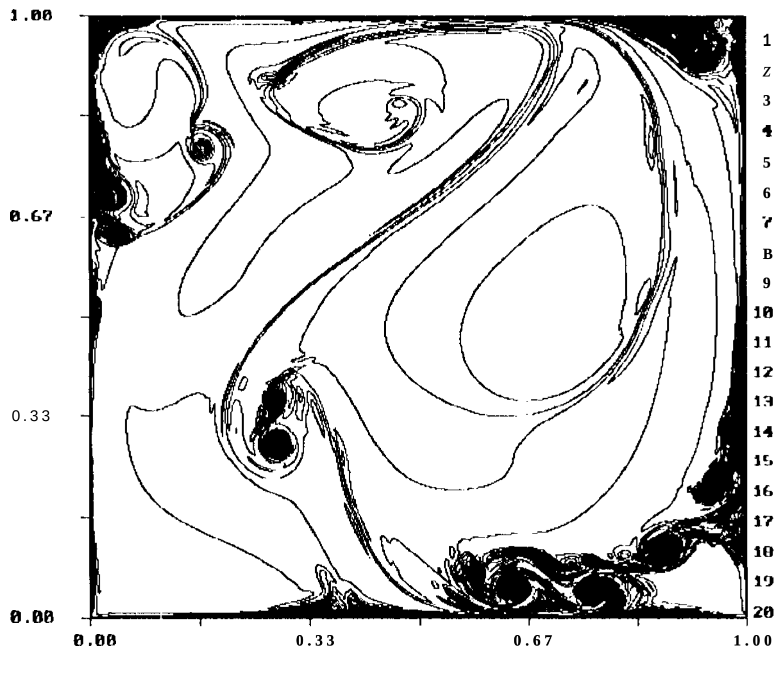


Figure 15: Velocity vector plot (top) and vorticity contour plot (bottom) from a driven-cavity flow with Re = 10^6, at time= 5.47. Grid size= 512x 512.

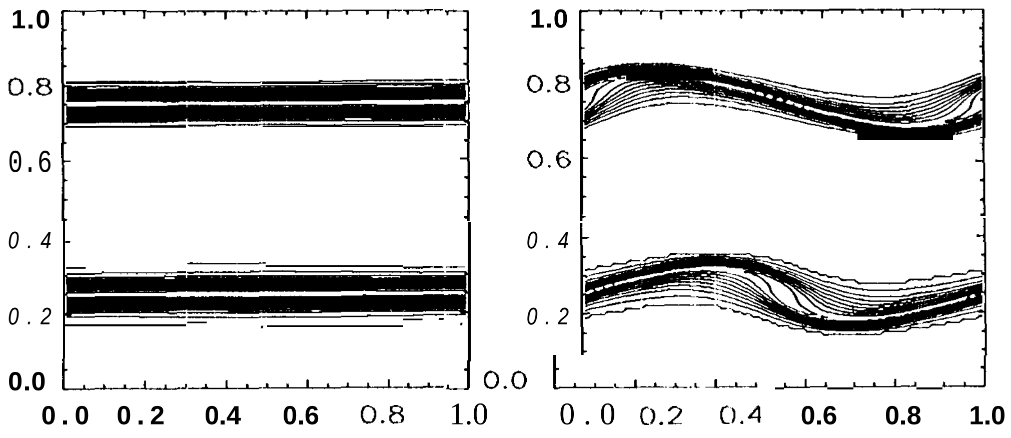


Figure 17: Vorticity contour plots from the periodic shear flow at time= 0.0 (left) and 0.62 (right). Grid size = 128 x 128.

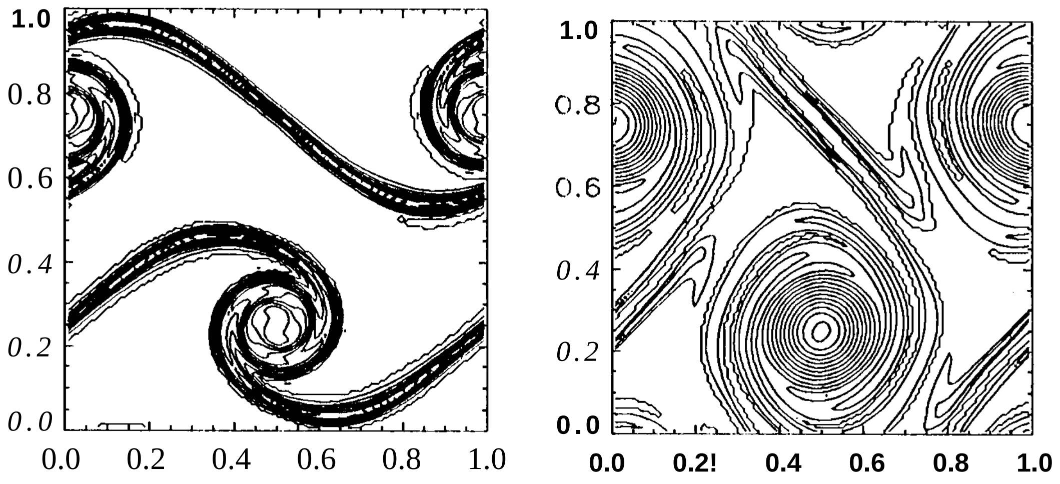


Figure 18: Vorticit y contour plots from the periodic shear flow for time= 1.25 (left) and time = 2.50 (right). Grid size= 128 x 128.

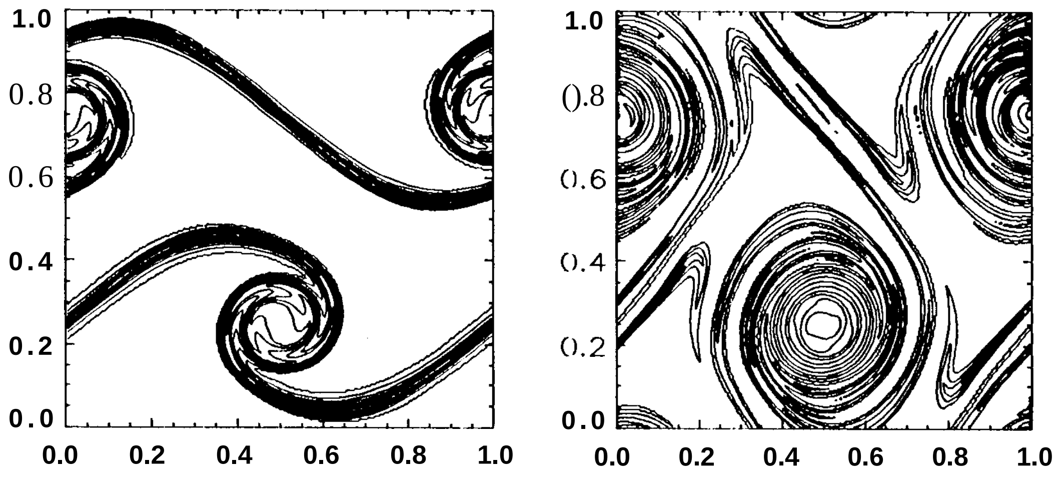
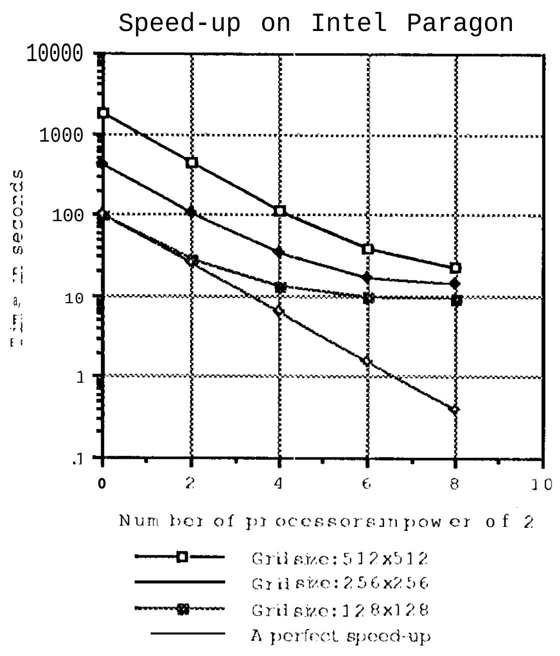


Figure 19: Vorticity contour plots from the periodic shear flow for time= 1.25 (left) and time =2.50 (right). Grid size = 256x 256.

A



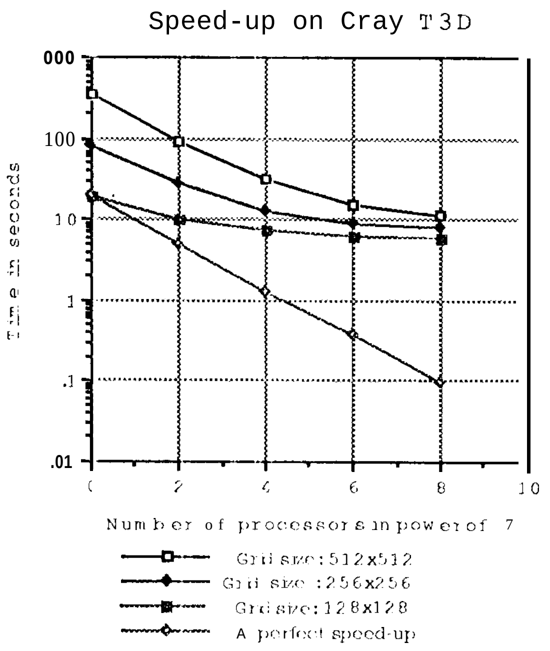
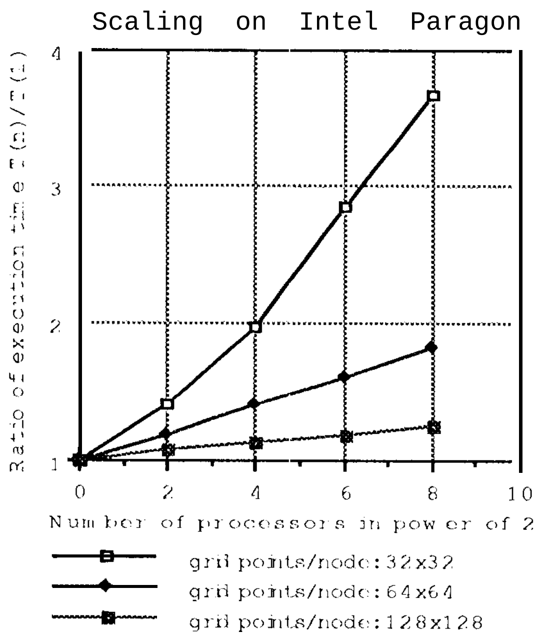


Figure 20: Speed-up performances of the parallel Navier-Stokes solver.



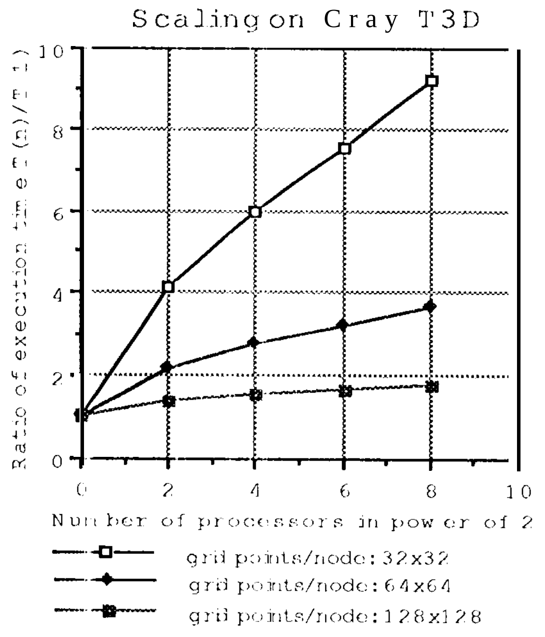


Figure 21: Scaling Performances of the parallel Navier-Stokes solver.